

**REMARKS**

Applicants respectfully requests further examination and reconsideration in view of the above amendments and the arguments below. Claims 1-19 are pending. Within the Office Action, Claims 1, 4, and 7-19 are rejected. Claims 2, 3, 5, and 6 are objected to. Claims 1, 8, and 10-13 are amended. Claims 1-19 now pending.

**Rejections Under 35 U.S.C. § 101:**

Within the Office Action, Claims 10-12 are rejected under 35 U.S.C. § 101 because the claimed invention is directed to non-statutory subject matter. Claims 10-12 are amended to constitute a machine within the meaning of 35 U.S.C. 101. For at least this reason, the independent Claims 10-12 are within allowable statutory subject matter.

**Rejections Under 35 U.S.C. § 102(b):**

Claims 1, 4, 7-9, 15, and 16 are rejected under 35 U.S. C. 102(b) as being anticipated by US Patent 5,778,349 granted to Okonogi, hereinafter referred to as "Okonogi." Okonogi teaches a means processing an input/output request for accessing an external device in a computer system with a plurality of subsystems (col. 1, lines 7-11). A request of data on an external device is partially processed on a subsystem and the remainder is processed on a second subsystem (col. 1, lines 27-31). In the example of Fig. 3, a user program is activated on a back end cluster (col. 3, lines 39-40). The back end cluster is loaded with a translation table to map logical addresses to an actual address of the main memory (col. 3, lines 40-47). When the program makes a system call to open a file, the back end cluster uses the translation table to translate the file offsets to block numbers (col. 3, line 60-61). The translations tables are used to translate parameter fields within a request. The interface by which the call is made, the system call interface 22, is not changed and thus the interface name does not change. Fields within a system call request are translated. Okonogi does not disclose the translation of the calling interface 22.

The independent Claim 1, as amended, is directed to a computer-assisted method for translating a logic module interface. The claim teaches the translation of one or more first interface names to a second interface having one or more second interface element names. In contrast to the teachings of Okonogi, the method utilized a translation of the calling interface where as Okonogi utilizes the same system calling interface to execute a logic module but where parameters within the call are translated. Thus, the interface being translated is a different limitation than translation of a parameter in a call. Therefor, Okonogi does not teach the limitations of Claim 1. For at least these reasons, the independent Claim 1 is allowable over the teachings of Okonogi.

The dependent Claim 4 is directed to adding an offset to the one or more interface element names. Claim 4 is dependent on the independent Claim 1. As described above, the independent Claim 1 is allowable over the teachings of Okonogi. Accordingly, Claim 4 is also allowable as being dependent on an allowable base claim.

The dependent Claim 7 is directed to logic module comprising an operating system where the first interface comprises a set of system calls to the operating system. Claim 7 is dependent on the independent Claim 1. As described above, the independent Claim 1 is allowable over the teachings of Okonogi. Accordingly, Claim 7 is also allowable as being dependent on an allowable base claim.

The independent Claim 8 is directed to a computer-assisted method of taking a first user module and generating a second user module where the provider interface is translated. Thus, a second executable user module is generated with a new and translated provider interface and therefore new and different code is generated for execution with a translated interface. Okonogi does not teach a modification to the executable code of changing the provider interface name. Okonogi teaches the taking of a parameter in a system call and modifying the parameter by using a translation table. Thus, Okonogi does not teach generation of a new module where the provider

interface is translated. Accordingly, Claim 8 is allowable over the teachings of Okonogi. For at least these reasons, the independent Claim 8 is allowable over the teachings of Okonogi.

The dependent Claim 9 is directed to a computer-assisted method of taking a first user module and generating a second user module where the provider interface is translated where the first provider interface comprises a set of system calls of an operating system, the second provider interface comprises a translated set of system calls of the operating system, the first user module comprises a software application referencing the first provider interface, and the second user module comprises a translated software application referencing the second provider interface. Claim 9 is dependent on the independent Claim 8. As described above, the independent Claim 8 is allowable over the teachings of Okonogi. Accordingly, Claim 9 is also allowable as being dependent on an allowable base claim.

The independent Claim 15 is directed to a method for processing a file access request comprising the steps of receiving a file access request having a first string and translating the first string to a second string indicating a file name, wherein the translating step proceeds according to a file name translation table. Okonogi teaches the mapping of a logical address, such as a file offset to an actual physical address (Okonogi: col. 3, lines 44-47). Thus, Okonogi does not teach the mapping of a first string to a second string. Accordingly, Claim 15 is allowable over the teachings of Okonogi. For at least these reasons, the independent Claim 15 is allowable over the teachings of Okonogi.

Claims 16 is rejected under 35 U.S. C. 102(b) as being anticipated by US Patent 6,275,938 granted to Bond *et al*, hereinafter referred to as “Bond.” Bond discloses an system and method for executing untrusted code including applets (Bond: col. 5, lines 23-28). Applet code is loaded into a predetermined memory area (Bond: col. 5, 28-30). During applet compilation, the emulator inserts memory sniff code into the compiled code (Bond: col. 5, 30-33). Some of the code is blocked by the emulator (Bond: col. 5, lines 42-50). Calls to an API are replaced with calls to “thunk code” where the API parameters are evaluated for harmful effects.

(Bond: col. 5, 51-63). Further Bond discloses a method of preventing an attack. The attack is performed by altering the data in a predetermined area of memory, the “sandbox”, that has been validated for passing call to an API. Another execution thread could alter the data being passed to the API in the time between the validation and the call and execution of the API. (Bond: col. 7, lines 59-66). An attack is prevented by copying the API arguments from the sandbox memory area to another location and copies a response to a memory area outside of where the applet can access the data (Bond: col. 7, line 67 - col. 8, line 11). Bond does not disclose the accessing of a file by changing a first filename extension to a second file name extension. Bond teaches the analysis of parameters in API calls by untrusted code for possible damaging effects.

The independent Claim 16 is directed to accessing a file with a first filename by changing a first filename extension to a second file name extension and accessing the file by the first filename with the second extension. As discussed above, Bond does not teach changing a parameter file name extension to obtain access. Bond teaches the validation of a parameter before making a API call or preventing an API call. Thus, Bond does not disclose the elements of Claim 16. For at least these reasons, the independent Claim 16 is allowable over the teachings of Bond.

**Rejections Under 35 U.S.C. § 102(e):**

Claims 1, 7-10, and 13 are rejected as being anticipated by Bond under 35 U.S.C. 102(e).

The independent Claim 1, as amended, is directed to a computer-assisted method for translating a logic module calling interface. The claim teaches a logic module where the calling interface is translated from one or more first interface names to a second interface having one or more second interface element names. The objective of translating the calling interface is to hide the interface from untrusted code which might guess at the calling interface name. In contrast to Claim 1, Bond teaches an unchanged calling interface name, the API, but the replacement of the API calling address code with “thunk code”, a modified API where the calling parameters can be

evaluated to determine if any system harm will occur by and actual the API call. Bond does not teach changing the calling interface name. Therefore, Bond does not teach Claim 1. For at least these reasons, the independent Claim 1 is allowable over the teachings of Bond.

The dependent Claim 7 is directed to a logic module where the calling interface is translated from one or more first interface names to a second interface having one or more second interface names, wherein the logic module is comprises an operating system and the first interface comprises a set of system calls of the operating system. Claim 7 is dependent on the independent Claim 1. As described above, the independent Claim 1 is allowable over the teachings of Bond (note, the Office action references Okonogi which is believe to be an unintentional error given that “thunk codes” are reference and are not found in Okonogi). Accordingly, Claim 7 is also allowable as being dependent on an allowable base claim.

The independent Claim 8 is directed to a computer-assisted method of taking a first user module and generating a second user module where the provider interface is translated. Thus, a second executable user module is generated with a new and translated provider interface and therefore new and different code is generated for execution with a different calling interface. Bond teaches the copying of executable code, an applet, to a “sandbox” and modifying the code to change the calls from the system API calls to the “thunk code” API calls. However, the calling interface name does not change for the application code calling the API. Thus, Bond does not teaches the taking a first user module and generating a second user module where the provider interface, the calling, interface is translated to a second interface. Accordingly, Claim 8 is allowable over the teachings of Bond. For at least these reasons, the independent Claim 8 is allowable over the teachings of Bond.

The dependent Claim 9 is directed to a computer-assisted method of taking a first user module and generating a second user module where the provider interface is translated where the first provider interface comprises a set of system calls of an operating system, the second provider interface comprises a translated set of system calls of the operating system, the first user module comprises a software application referencing the first provider interface, and the second

user module comprises a translated software application referencing the second provider interface. Claim 9 is dependent on the independent Claim 8. As described above, the independent Claim 8 is allowable over the teachings of Bond. Accordingly, Claim 9 is also allowable as being dependent on an allowable base claim.

The independent Claim 10 is directed to a computer system comprising an operating system, a processing module running the operating system with a set of system calls to the operating system where a first set of system calls are for trusted software modules and the second set of dummy system calls are for trapping untrusted software modules. In contrast to the Applicant's claim, Bond disclosed an operating system where the system calls, to the API's for example, are copied to a sand box and checked to determine if the API call could cause any harm to the system. Bond only teaches one API interface where the application making the call is modified to first go to "Thunk code" to determine if the harmfulness of the code. If the API call is deemed not to be harmful, the API call is not trapped as in Claim 10, but is passed to the real API call. Thus Bond differs in two ways. First the calling interface name does not change. Secondly there are not dummy system calls to trap untrusted code. For at least these reasons, the independent Claim 10 is allowable over the teachings of Bond.

The independent Claim 13 is directed to a hardware processing system comprising a processor; an instruction translation table; a module translation table, where the processor fetches a instruction, decodes the instructions according to the instruction translation table, executes the instruction and wherein instructions that reference calls to modules are translated according to a module translation table. In contrast to the Bond, the Applicant's claim discloses a translation of a module name called by the instructions. Bond teaches one API interface where the application making the call is modified to first go to "Thunk code" to determine if the harmfulness of the code. If the API call is deemed not to be harmful, the API call is not trapped, but is passed to the real API call. Therefore, Bond does not teach the translation of a called module name. Thus, Bond does not teach all of the limitations of Claim 13. For at least these reasons, the independent Claim 13 is allowable over the teachings of Bond.

Claims 14, and 17-19 are rejected under 35 U.S.C. 102(e) as being anticipated by US patent application 2004/0255161 granted to Cavanaugh, hereinafter referred to as "Cavanaugh." Cavanaugh teaches a means systems and methods for examining communication streams to identify and eliminate malicious code (Cavanaugh: Abstract). Cavanaugh discloses the examination of a packet to determine whether a packet is passed or blocked. (Cavanaugh: [0052]). Cavanaugh teaches the removal of malicious code or messages by the removal of packets or streams (Cavanaugh: [0062]). Cavanaugh does not teach the translation of a web page request from a first URL to a second URL according to a URL translation table, the generation of a second web page where the URL in the first web page is translated to third URLs in the second web page.

The Claim 14 is directed to receiving a web page request having a first URL, translating the first URL to a second URL according to a URL translation table. The second URL indicating a first web page, the first web page having a third embedded URLs. A second web page is generated wherein the generating step comprises replacing the third embedded URLs with translated versions of said third embedded URLs according to the URL translation table to obtain the second web page. Cavanaugh does not disclose such translations of URL in web pages. Thus, Cavanaugh does not disclose the above claim. Accordingly, Claim 14 is allowable over the teachings of Cavanaugh. For at least these reasons, the independent Claim 14 is allowable over the teachings of Cavanaugh.

The Claim 17 is directed at processing a network connection request. The connection request has a first port number where the first port number is translated to a second port number according to a translation table. As discussed above, Cavanaugh examines packets to determine if the packet should be passed or blocked (Cavanaugh: [0052]). Cavanaugh does not teach the translation of packet port numbers for a network connection request. Thus, Cavanaugh does not disclose the above claim. Accordingly, Claim 17 is allowable over the teachings of Cavanaugh. For at least these reasons, the independent Claim 17 is allowable over the teachings of Cavanaugh.

The Claim 18 is directed at receiving a network packet. The network packet has a protocol field having a first identifier. The first identifier is translated to a second identifier according to a protocol type translation table. As discussed above, Cavanaugh examines packets to determine if the packet should be passed or blocked (Cavanaugh: [0052]). Cavanaugh does not teach the changing or translation of packet protocol fields. Thus, Cavanaugh does not disclose the above claim 18. Accordingly, Claim 18 is allowable over the teachings of Cavanaugh. For at least these reasons, the independent Claim 18 is allowable over the teachings of Cavanaugh.

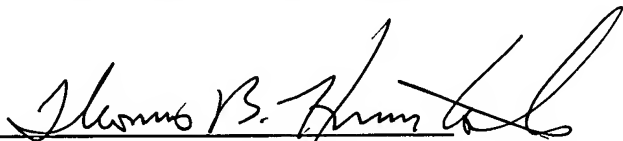
The Claim 19 is directed at a method of processing a database command. The database command is comprised of a first string where the first string is translated to a second string according to a database keyword translation table. As discussed above, Cavanaugh examines packets to determine if the packet should be passed or blocked (Cavanaugh: [0052]). Cavanaugh does not mention databases. Cavanaugh does not mention strings or the translation of strings. Thus, Cavanaugh does not disclose the above claim. Accordingly, Claim 19 is allowable over the teachings of Cavanaugh. For at least these reasons, the independent Claim 19 is allowable over the teachings of Cavanaugh.

Applicants respectfully submit that the claims are in a condition for allowance, and allowance at an early date would be appreciated. Should the Examiner have any questions or comments, they are encouraged to call the undersigned at (408) 530-9700 to discuss the same so that any outstanding issues can be expeditiously resolved.

Respectfully submitted,

HAVERSTOCK & OWENS LLP

Dated: 2-15-08

By:   
Thomas B. Haverstock  
Reg. No. 32,571  
Attorneys for Applicant

**CERTIFICATE OF MAILING (37 CFR 1.6(c))**

I hereby certify that this paper (along with any referred to as being attached or enclosed) is being deposited with the U.S. Postal Service on the date shown below with sufficient postage as first class mail in an envelope addressed to the: Commissioner for Patents, P.O. Box 1450 Alexandria, VA 22313-1450

HAVERSTOCK & OWENS LLP

Date: 2/15/08 Per: 